Building Hybrid Rover Models for NASA: Lessons Learned

Thomas Willeke[1] and **Richard Dearden**[2]

Abstract. Particle filters have recently become popular for diagnosis and monitoring of hybrid systems. In this paper we describe our experiences using particle filters on a real diagnosis problem, the NASA Ames Research Center's K-9 rover. As well as the challenge of modelling the dynamics of the system, there are two major issues in applying a particle filter to such a model. The first is the asynchronous nature of the system—observations from different subsystems arrive at different rates, and occasionally out of order, leading to large amounts of uncertainty in the state of the system. The second issue is data interpretation. The particle filter produces a probability distribution over the state of the system, from which summary statistics that can be used for control or higher-level diagnosis must be extracted. We describe our approaches to both these problems, as well as other modelling issues that arose in this domain.

1 Introduction

Diagnosis is of great importance for many current and planned NASA missions. Unfortunately, classical approaches such as Livingstone [10] are very hard to apply to many of the systems currently being deployed, or planned for future missions. In particular, attempts to apply these approaches to planetary rovers have been notably unsuccessful, due to the considerable interaction between a rover and its environment, and the consequent difficulty in discretizing the observations, and producing a discrete rover model—as required by Livingstone—that is capable of making useful diagnoses.

Over the past several years we have been developing algorithms to perform hybrid diagnosis on-board a rover. Since a discrete model appears to be impractical, we have developed new models using a hybrid discrete-continuous representation of the rover, and applied hybrid diagnosis algorithms based on particle filters[3] to them. This paper reports on the complexities of the modelling task, and some lessons we have learned while making our first high-fidelity models of the rover behaviour and testing standard particle filtering algorithms on them.

Figure 1 shows the K-9 rover testbed at NASA Ames Research Center, on which these experiments were run. K-9 is a six-wheeled rover of the same class as the MER rovers currently exploring Mars, although with considerably more on-board processing power. It has a number of subsystems including the locomotion system (the wheels, suspension, driving and steering motors), an instrument arm (visible folded up under the solar panels on the right hand side in the figure), the pan-tilt head and cameras (top of figure), and the power subsystem. Eventually we plan to have a model of all of these, but at present the model only looks at the locomotion system, and receives sensor data from the wheels and the suspension (shown in detail in Figure 3).

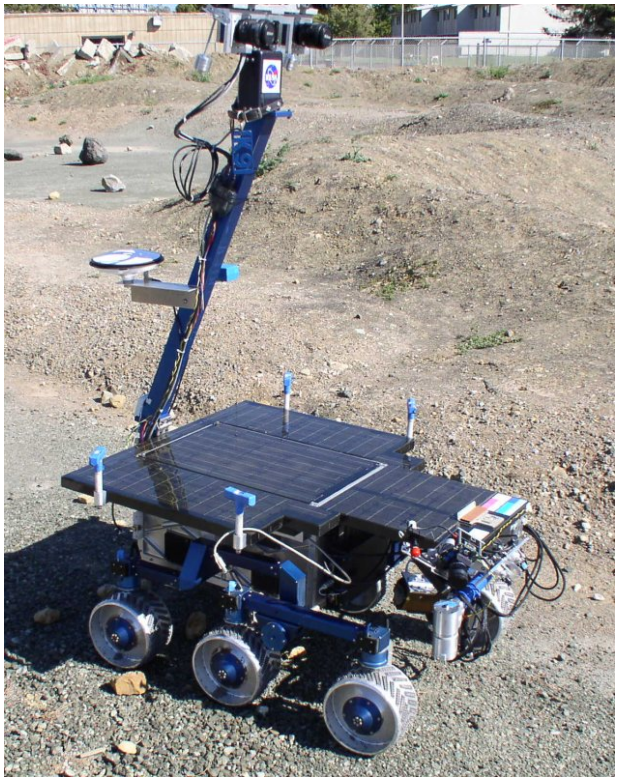

Figure 1. The K-9 rover.

Figure 2 shows the standard particle filter algorithm that we will discuss in this paper. While more sophisticated variants such as Rao-Blackwellized particle filters [2, 5] and the Gaussian particle filter [4] have also been applied to rover diagnosis [1], they currently don't handle the continuous-time approach we have found it necessary to adopt for this model (see Section 3). The algorithm consists of three main steps that are performed at each time-step. The first (Step 3.(a)i and ii in the Figure) is the *Monte Carlo step*, where each sample is projected into a possible future state in a stochastic manner. Following this is the *re-weighting step* (Step 3.(a)iii), in which we condition on the observations of the new state by re-weighting each of the samples by how likely it is that the observation could be generated from the state represented by that sample. Finally in the *resampling step*

[1] QSS Group Inc. / NASA Ames Research Center, M.S. 269-3, Moffett Field, CA 94035-1000 USA. *twilleke@email.arc.nasa.gov*
[2] RIACS / NASA Ames Research Center, M.S. 269-3, Moffett Field, CA 94035-1000 USA. *dearden@email.arc.nasa.gov*

(Step 3.(b)), new samples are created by sampling from distribution induced by the weighted samples—the new samples are all copies of the old ones, and the probability of each sample being copied is proportional to its weight.

1. For N particles $p^{(i)}$, $i = 1, \ldots, N$, sample discrete modes $z_0^{(i)}$, from the prior $P(Z_0)$.
2. For each particle $p^{(i)}$, sample $x_0^{(i)}$ from the prior $P(X_0|z_0^{(i)})$.
3. for each time-step t do
   (a) For each particle $p^{(i)} = (z_{t-1}^{(i)}, x(i)_{t-1})$ do
      i. Sample a new mode: $\hat{z}_t^{(i)} \sim P(Z_t|z_{t-1}^{(i)})$.
      ii. Sample new continuous parameters: $\hat{x}_t^{(i)} \sim P(X_t|\hat{z}_t^{(i)}, x_{t-1}^{(i)})$.
      iii. Compute the weight of particle $\hat{p}^{(i)}$: $w_t^{(i)} \leftarrow P(y_t|\hat{z}_t^{(i)}, \hat{x}_t^{(i)})$.
   (b) Resample N new samples $p^{(i)}$ where: $P(p^{(i)} = \hat{p}^{(k)}) \propto w_t^{(k)}$

Figure 2. The particle filtering algorithm.

While the algorithm is conceptually simple, building a suitable hybrid model of the rover, and applying the algorithm to the model both turn out to be quite complex tasks. Although we won't discuss it here in detail, finding suitable differential equations to describe the system behaviour is very time-consuming and difficult. More significantly, the realities of applying diagnosis to the rover necessitated many changes both to the model, and to the way the algorithm was applied.

The software architecture used on-board the rover is CLARAty [7], which is a hierarchical architecture under which subsystems are kept as independent as possible, and communicate with one another only by messages travelling up and back down the hierarchy. In this respect, CLARAty seems quite typical of generic robotics architectures currently in development. However, this has two significant effects for a diagnosis system. Firstly, the system must integrate data from sensors in different subsystems, and it is quite possible for these sensors to deliver data at different rates, and secondly, the system must provide diagnosis data for subsystems that can be passed up the CLARAty hierarchy to be used for control decisions, or even potentially as inputs to other, higher-level diagnosis algorithms. The first of these issues necessitates the use of a continuous-time model and methods to handle asynchronously arriving data. We discuss the implications of this in Section 3. The second, along with the fact that particle filters produce as their output probability distributions over the states of the system, requires novel ways to summarise the output from the filter, as the full probability distribution is far too large to reasonably deliver to either a control algorithm or a more abstract diagnosis algorithm as an input. We discuss possible solutions to this problem in Section 4.

2 Model Overview

Our long term goal for this model is to be able to track many different faults across multiple subsystems of the rover. To that end, we have started by building a model that tracks normal driving over different terrain and tracks one class of faults. Currently, the model has 18 binary discrete variables and 30 continuous variables: Each of the six wheels has 3 binary discrete state variables: DRIVING, TERRAIN, and SPEEDSENSOR. Each wheel also has three continuous variables: BASELINE, HEIGHT, and SPEED, and two observable variables: OBS.SPEED and OBS.HEIGHT.

Figure 3. The suspension on the K-9 rover. The front of the rover is on the left, wit the first two wheels attached to the bogey, which is in turn attached to the rocker, and then to the rover chassis.

The continuous variables capture the basic motion of the rover. Before understanding the parameters themselves there are a some aspects of the rover that must be discussed. The rover employs a rocker-bogey suspension system for its six wheels, as shown in Figure 3. The two front wheels on either side are attached to the bogey, which swings freely around its centre, where it is attached to the rocker. The rocker in turn can pivot around the differential axle which is attached to the rover chassis. The only position information we have about the wheels comes from the two bogey angles (the angle between the bogey and the rocker) and the differential axle (there are also steer angles for each wheel that we plan to add to the model, but are currently ignored). The differential axle guarantees that the chassis of the rover will always be centred between the wheels, and thus these three angles are sufficient data to calculate the heights of the wheels relative to the chassis of the rover. There are two difficulties to consider. First, due to the differential axle, there is an ambiguity between situations where the rover has its left wheels high on a rock versus its right wheels down in a hole. Looking only at the angle information, these situations look exactly the same, and ultimately will require integrating inertial information to disambiguate. The other difficulty is that a slight calibration error or measurement noise in the differential axle at the rear of the robot can by magnified by the long lever arm of the rocker and bogey structures to create a substantial change in measured heights at the front wheels. We found that every time we tested the rover, and looked at the wheel heights, they seemed substantially different. We postulated that there were slight calibration errors in the differential axle each day caused by lifting and transporting the rover from the lab to the simulation mars yard. As a result of this calibration error and resulting large change in measured wheel heights, we could not use simple threshold values on the height of the wheels to categorise their position or the terrain.

With these issues in mind we designed the model to be self cali-

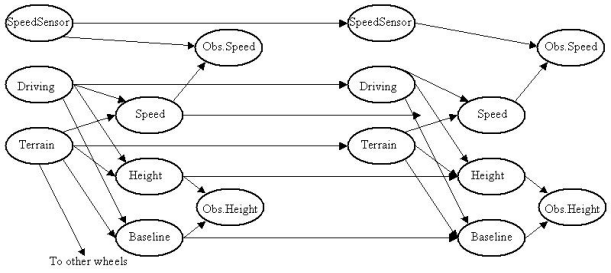

Figure 4. Dynamic Bayesian network representation of the current rover model for a single wheel. The variables on the left represent the state at time t, and on the right time $t + 1$.

brating. The BASELINE parameter is the calibration of the wheel for each run, giving an approximate idea of where the wheels neutral "zero" position should be in relationship to the chassis of the robot. As a result, the HEIGHT parameter, which is the wheels height above (or below) that BASELINE, is a much better behaved (calibration independent) variable and can be used to infer information about the terrain that the rover is traversing. Finally, the SPEED parameter is simply the speed of the wheel. The SPEED parameter is used as a multiplier in the differential equations governing the HEIGHT parameter. This follows from the intuition that the faster the rover is moving the greater the potential change in the height will be as it is capable of climbing a larger rock in the same amount of time. It is also worth mentioning that the speed of the wheel is not the same as the overall speed of the rover. When the rover turns the speeds of wheels on opposite sides will be different, or possibly even completely reversed if the rover is executing a point turn.

The model contains two different types of discrete variable. The DRIVING and SPEEDSENSOR variables represent operational modes of the rover while the TERRAIN variable is a somewhat arbitrarily defined property of the environment that is added to aid in modelling the system. The DRIVING variable tells us if the wheel is stopped or driving. As well as representing a commanded rover mode, it also prevents the BASELINE parameter value from drifting to match the current height of a wheel when the rover is parked on top of a rock. To achieve this, the differential equations in the stopped mode do not allow the BASELINE parameter to change. The TERRAIN variable tells us if the robot is traversing reasonably flat ground, or rocky terrain. This distinction is useful because the high volatility of height values when the rover is traversing rocky ground makes it desirable to limit how closely the parameters match the observed data. One does not want the baseline, a critical calibration parameter, to drift off and track a transitory signal caused by the rocky terrain. Finally, the SPEEDSENSOR failure state exists to capture the situation where the rover is reporting a speed of zero yet is also reporting that its wheel heights are changing. We have seen examples of this in the data generated by the rover, and since the speed of the rover is used in the model, we need to track those moments when the speed is being inaccurately reported.

Figure 4 shows a dynamic Bayesian network representation of the current model for a single wheel. As the figure shows, there are lots of interactions between the variables inside the wheel model, but relatively few between variables in different wheels. While the number of these will certainly grow as the model gets more complex, at present we can treat the wheels almost independently, which considerably improves the efficiency of the model by reducing the dimensionality of the state space.

One failure that we plan to add to the model in the near future, and that will require more connections between the wheel variables is fondly referred to as "the Rover Rampant" failure. Under some poorly understood circumstances the centre wheels drive quicker than the front wheels and end up pushing the front of the bogey up into the air, with the result that the rover looks like the stylised lion with its paws in the air from a medieval knights shield. This is a serious fault with a rocker-bogey rover that has occurred in field trials. To diagnose this will take a more general view of the whole rover since each wheel is behaving correctly when taken in isolation, but data from a number of wheels over time can be combined to detect the fault.

3 Asynchronously Arriving Data

Traditionally, particle filters, as with most other filtering algorithms, are viewed as discrete time step algorithms. This means that the whole system moves forward in discrete chunks of time, with new data arriving, the model updating by one cycle, and the process repeating. While this is conceptually easy to handle, it is unfortunately at odds with the design of many robotic systems. The world itself is asynchronous and robots often are too. One reason being that they usually have many different sensors for measuring their environment, and each of those sensors will have different update frequencies which are dictated by their hardware. In our case, the K9 rover is such an asynchronous rover, with the further complication that data is sent from the control process to the diagnosis process over a CORBA link. CORBA is a message passing service which is often used in modular systems designs so that separate components (such as the diagnosis and control modules) do not need to depend upon each other and can be compiled as completely separate programs (a useful feature when developing complex projects with large distributed teams). However, this means that there are a number of steps between a sensor making a measurement, the measurement getting recorded and time stamped, the packet being pushed into the CORBA link, and our diagnosis engine receiving the data and processing it. As a consequence, data arrives at variable rates, at different times from different rover subsystems, and does not always arrive in proper temporal order. While data from a single source has no trouble arriving in sequence, sometimes we will get data from source A with a time stamp of X and then get data from source B with a time just slightly before X.

Given this characteristic of the data, the diagnosis system needs to be able to handle data that arrives in an asynchronous fashion. One naive way to handle this is to keep the diagnosis system as a discrete time step algorithm and wait until data from each source has arrived, and then to run the diagnosis system though one step. This only works if the data from each source is produced at a similar frequency. If the frequencies are very different, do you wait for the slow one, throwing away other data? What happens when one sensor fails and stops producing any data? How long do you wait for the data before giving up? Ultimately, whatever approach is used, the most importantly question is, how do you run in real-time? What happens if processing slows down and the system is handling data slower than it is produced? Or, what does one do when the system is running fast enough to handle all the data produced if it arrived evenly spaced, but instead data from two different sensors arrives almost simultaneously followed by a long quiet spell?

We built our system with these real-time asynchronous questions in mind. There are two branches of the real-time asynchronous data problem: computational and algorithmic. The computational aspect

refers to all the problems of receiving and controlling the flow of data, and ensuring that the real-time constraint is always met. The algorithmic aspect refers to changes that occur to the model and the particle filter algorithm in response to the introduction of asynchronous data.

In order to handle the computational complexities we pass the arriving data to the InputModel. The InputModel can be run in buffered mode where it will guarantee that every piece of data will be processed, but usually it is run in real-time mode. In this case, arriving data is put into a separate bin for each sensor type and is labelled as NewData. If there was data still in that bin, it simply gets overwritten – a sad loss, but necessary to ensure that the model does not fall behind in processing the data produced by the rover. The particle filter, in a separate thread, is constantly going sequentially through these bins looking for NewData and processing it when it finds it (and removing the NewData label).

The more interesting changes happen when dealing with the algorithmic aspects of asynchronous data. Our primary changes were to parameterise the model on time and to perform forward prediction based on the marginals probabilities of the observed sensor. These changes have many interesting consequences which will be covered in detail.

Parameterising on time involves computing the amount of time between the new data and the last time the model was updated. This value is then used as a scaling factor so that the model changes by some amount proportional to the amount of time that has passed. Unlike Kalman filters and other tracking algorithms, one major advantage of the standard particle filter is that it can be run relatively easily on a continuous-time model like this. If we moved to a more sophisticated variant such as Rao-Blackwellized particle filters [1] or the Gaussian particle filter [4], we would have to integrate the model effects over time using a continuous-time Kalman filter or some similar approach.

3.1 Conditioning on the Observations

Once the model has been synchronised with the timestamp of the data we condition on the observations by re-weighting the samples as shown in Step 3.(a)iii. of the algorithm in Figure 2. In a traditional discrete-time particle filter one would have observations for all the observable variables at this point and would simply calculate the weight of the sample based on the full probability of the observations. In our case, we calculate the weight of the sample based on the marginal probability of the available observed variables.

This change to re-weighting based on the marginal probabilities has a number of important consequences. While one parameter is being re-weighted by the observations, all the other parameters are free to drift around according to the dynamics of the model until there is an observation of them later. Figure 5 shows an example of this effect. The probability that the rover is stopped tends to change as observations of speed and wheel height alternately arrive. When an observation of wheel height arrives, samples naturally change from STOPPED to DRIVING or vice-versa due to the stochasticity in the model, and since the distribution that results from the Monte Carlo step of the particle filter is then marginalised on the OBS.HEIGHT variable, these samples seem quite plausible. When a subsequent observation of speed arrives, these samples get low weight because they predict that variable poorly.

Given that our choice of approaches to asynchronous data has led to this problem, it is worth looking back and seeing if there are ways to avoid it. One alternative is to keep the algorithm as a time discrete system and use the most recent observations from each sensor, even if some of those observations have not updated for a while (this amounts to padding the data with old observations). This is appealing, and would produce output that superficially looked cleaner since it would not have the zigzag effect. Unfortunately, the consequences of this are that you are forcing the model to continue to match the last observation of the parameter in question, thus a majority of the samples will have values close to that last observation. In reality the parameter is likely changing over this new time, and when the new data does arrive, most of the samples will be clustered around the old data, meaning that fewer of them will match well to the newly arrived data. On the other hand, with our method, the model keeps progressing even if new data isn't arriving for a while. Thus, the samples will start to spread out into a growing error cloud, keeping them well distributed in the range of possible observations. Thus, when the new observation arrives, there is a much higher likelihood that there is a sample close to the new observation.

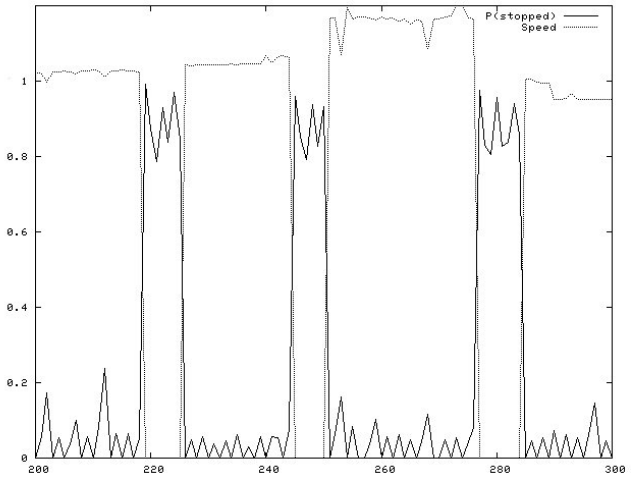

Figure 5. Wheel speed over time for one of the wheels compared with the probability that the wheel is stopped. The probability varies from step to step because when there is no observation from the speed sensor samples tend to move from the STOPPED to the DRIVING state (or vice-versa). When a new observation arrives, the samples in the "wrong" state die out.

Another approach would be to only update (i.e. forward predict) the parts of the model for which data is arriving. This has the appeal that the model doesn't drift around much, instead only advancing those parts of it that are seeing new data. But this results in a model that is fractured across time. Since the purpose of the model is to give a computational estimate of the current state of the rover, having the model fractured back across time by different amounts for each parameter would make it almost impossible to create any coherent view of the state of the rover.

3.2 Frequency of Observations

So far in our discussion of asynchronous data arrival, we have generally been assuming that, within a certain bound, the frequency of each individual sensor was fairly constant. What happens if we relax this constraint and allow the frequency at which each sensor produces data to vary widely, perhaps based on different operational modes of the rover? It turns out that the K-9 rover does exactly this. When K-9 is moving, it produces information about its speed at a regular fre-

quency, but when K-9 is stopped it produces speed data at a much lower frequency (about a factor of 20 slower). This behaviour makes sense since there is not much happening worth reporting as far as the speed goes when the rover is stopped. But meanwhile, the height data keeps arriving at its usual frequency. This ends up being a problem since the model keeps updating and, since there is nothing to pin the speed values down, the model speed starts drifting away.

In some respects, there may be no good generalised solution for this sort of problem: how can a model stay accurate if it simply is not getting any data? Yet, in this case we were able to make some safe assumptions. Since the rover always reports its speed data when it is moving, it is safe to assume that if we have seen the rover stop, and have not seen any new data about its speed, it is probably still stopped. Following this logic, when we have seen the rover stop, and then don't get any fresh speed data for a while, we start inserting fake (stopped) observations. This allows the model to nail down its speed values and stop the drifting parameter.

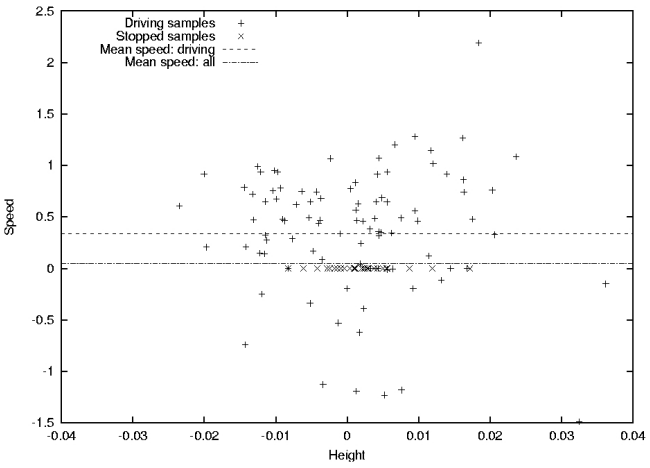

Figure 6. Height graphed against Speed for all the samples. Samples in the driving state are distinguished from those in the stopped state. The graph shows the bimodal distribution of speed due to the discrete states.

3.3 Out of Order Data

We said above that we occasionally get data out of order. This is a potential problem for any model. Fortunately, the data process on K-9 is relatively fast, so data that arrives out of order is generally only a few milliseconds old. Since we would like to include these observations in our model, we have come up with three methods to handle this problem. The first is simply to run the model with a small delay, long enough to ensure that all previous data has arrived. This works well most of the time, but leads to a small delay in producing a diagnosis. It can also lead to problems when data arrives very close together from different subsystems—the diagnosis system receives new data while it is waiting, and then must wait again to ensure everything is ordered correctly. In practice this tends to resolve itself quickly, but the potential for the system to block indefinitely is there, and the likelihood will increase as more data from more subsystems is incorporated into the model.

The second approach we looked at is to simply run the model backwards in time. Since the differential equations are all time-parameterised, this is relatively simple to do, although it may get more difficult as the model complexity increases. It results in a state estimate that is slightly older than the most recent observation, despite the fact that that observation has been incorporated, so it isn't an entirely accurate estimate of the current state. The opposite problem is encountered in our third approach, to simply change the time-stamp on the out of order data to match the current time, and incorporate it. Again, this is relatively straightforward to implement, but produces a small error in the resulting state estimate. In practice, these last two approaches tend to give very similar results, due to the very small differences between the times of the out of order data.

4 Data Interpretation

The point of any diagnosis system is to provide information about the state or states the system is in at any moment. For a particle filter, as with any approximation to Bayesian belief updating, this state information will be in the form of a probability distribution over the possible states. For the discrete modes, this is fairly straightforward. The probability that the system is in some state s is simply the weighted sum of all the samples in s divided by the weighted sum of all the samples. However, for the continuous system parameters, this is not quite so simple. The problem is that this is the marginal distribution of a particular variable. To see why this is a problem, consider the driving speed of a single wheel. In a state where some of the samples are in the stopped state, and others are in driving states, the speed of the stopped samples is zero, while the speed of the driving samples is much larger. Effectively the speed has a bimodal distribution, as shown in Figure 6. If we simply compute the mean speed over all the samples (taking the marginal distribution over speed), we get an estimate of the speed of the wheel that is much lower than the mean of the samples that actually have the wheel driving. In Figure 6 this gives a mean speed of 0.05, compared with 0.34 for the samples in the driving states alone.

The naive solution to this problem is to report the entire joint probability distribution as the output of the diagnosis algorithm. Unfortunately, this consists of the complete set of particles, and is clearly too large to be practical. We need to select some appropriate summary statistics to report, but the question is what is most useful.

We have looked at three main approaches. The first is simply to report the median rather than the mean for the value of the continuous variables. This actually works quite well for the example here (it reports a speed of zero since most of the samples are in the STOPPED state), but is rather unsatisfactory in general. One problem is that if two states have approximately equal probability, the system may flip between them, reporting a value first from one and then from the other. A second problem is that we lose access to variance information so we can't tell how confident the diagnosis is in its estimate of a continuous parameter.

The other two approaches involve computing marginals over the discrete states. In the first, we simply report the marginal mean and variance for each discrete state in the system. For example, at the time shown in Figure 6, the following table would result (only a single wheel, only the speed data, and no variance information is shown):

Terrain	Driving	Sensor	#samples	Mean speed
flat	stopped	fault	8	0
flat	stopped	OK	412	0
flat	driving	fault	3	0
flat	driving	OK	82	0.33
rocky	stopped	fault	21	0
rocky	stopped	OK	419	0
rocky	driving	fault	3	0
rocky	driving	OK	52	0.38

This is probably more information than is really needed, especially as there are 2^n rows in this table for n discrete variables. A more compact result is the $2n$-row table that results from computing the marginal for each discrete variable individually. In this case we get:

Terrain	#samples	Mean speed
flat	505	0.05
rocky	495	0.04
Driving	#samples	Mean speed
stopped	860	0
driving	140	0.34
Sensor	#samples	Mean speed
fault	35	0
OK	965	0.05

While this provides less information than the previous approach, it seems to capture the essence of the data, at least in this example, and scales more reasonably as the model grows. Furthermore, the structure of the Bayesian network in Figure 4 shows which marginals are required, and may allow us to reduce the amount of data produced by leaving out marginals for variables that are unrelated.

5 Conclusions and Future Work

This paper presents a number of problems that we have encountered in building a particle filter-based diagnosis system for a planetary rover. In particular we discuss problems with model building, coping with data that arrives asynchronously from sensors in different rover subsystems, and potentially out of temporal order as well, and with representing the output of the particle filter sufficiently compactly for other algorithms to use. We offer possible solutions based on our experiences to a number of these problems, although the solutions we have adopted so far may not scale to larger models, or may be superceded as our study of the problem continues.

One of the great advantages of the model-based discrete diagnosis systems is the compositionality of their models. While some progress has been made on building compositional models of hybrid systems [6], and exploiting that structure in algorithms [8], there is much still to be done to make these tools easy to use and effective in practice.

A second major problem as the model gets more complex is parameter tuning. Automated tools for building hybrid models are desperately needed, both user-interface type tools for describing the models component by component, and linking them together, and tools for automatically tuning the parameters of the model to best match the data. For this second need, we are hoping for progress in quantitative model learning (for example, see [9]), possibly applied in a supervised manner, where a user will pick out a set of data produced in a single system mode, and the learning algorithm will then produce a model that matches that data.

Finally, the model we have described is still relatively preliminary, and we are actively adding both new fault modes, and new sources of data and rover subsystems to it. One of the consequences of this is that the number of samples needed to effectively estimate the current state will continue to increase, leading to computational issues. The obvious solution is to move to a more expressive representation such as Rao-Blackwellized particle filters[2], or their non-linear variants, which can significantly reduce the number of samples needed to effectively represent the current belief state, and hence the computational requirements of the algorithm. However, as we discussed in Section 3, this leads to a number of problems with the continuous-time nature of the model. Adapting these algorithms to use continuous-time Kalman filters, or similar approaches will become more critical as the model grows. In our opinion, this is the most challenging problem we face as we continue modelling the rover.

Acknowledgements

Thanks to the K-9 team at NASA Ames Research Center for their efforts to make K-9 available as a test-bed for our diagnosis algorithms, and for their support in integrating the algorithms into the CLARAty architecture. This research is supported by funding from NASA's Mars Technology Program, and the Intelligent Systems program.

REFERENCES

[1] Nando de Freitas, Richard Dearden, Frank Hutter, Ruben Morales-Menendez, Jim Mutch, and David Poole, 'Diagnosis by a waiter and a mars explorer', *Invited paper for Proceedings of the IEEE, special issue on sequential state estimation*, (2003).

[2] Arnaud Doucet, Nando de Freitas, Kevin Murphy, and Stuart Russell, 'Rao-blackwellised particle filtering for dynamic bayesian networks', in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 176–183, Stanford, (2000).

[3] *Sequential Monte Carlo in Practice*, eds., Arnaud Doucet, Nando De Freitas, and Neil Gordon, Springer-Verlag, 2001.

[4] Frank Hutter and Richard Dearden, 'The gaussian particle filter for diagnosis of non-linear systems', in *Proceedings of the Fourteenth International Workshop on the Principles of Diagnosis*, Washington, DC, (2003).

[5] Ruben Morales-Menendez, Nando de Freitas, and David Poole, 'Real-time monitoring of complex industrial processes with particle filters', in *Neural Information Processing Systems (NIPS)*, (2002).

[6] Sriram Narasimhan, *Model-based Diagnosis of Hybrid Systems*, Ph.D. dissertation, Vanderbilt University, Nashville, TN, USA, August 2002.

[7] I.A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and Won Soo Kim, 'Claraty: An architecture for reusable robotic software', in *SPIE Aerosense Conference*, Orlando, Florida, (April 2003).

[8] Brenda Ng, Leonid Peshkin, and Avi Pfeffer, 'Factored particles for scalable monitoring', in *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, Edmonton, (2002).

[9] Ljupčo Todorovski, Sašo Džeroski, Ashwin Srinivasan, Jonathan Whiteley, and David Gavaghan, 'Discovering the structure of partial differential equations from example behavior', in *Proc. 17th International Conf. on Machine Learning*, pp. 991–998. Morgan Kaufmann, San Francisco, CA, (2000).

[10] Brian C. Williams and P. Pandurang Nayak, 'A model-based approach to reactive self-configuring systems', in *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference*, pp. 971–978, Portland, Oregon, (1996). AAAI Press / The MIT Press.